Lab1实验报告

一.设计思路

a)L版本设计思路

进行二进制乘法a*b，最简单的思路是将乘法看作若干次加法，只需要将其中一个数b看作另一个数a要累加的次数，另一个数b累加a次即可得到答案。

这种思路非常简单，代码量极小，适合做L版本。

但是一遍一遍累加要执行的次数完全取决于b的大小，若b为4000，该程序就要连续执行4000次，显然是我们不想看到的。

$$result = a + a + a + \ldots (b个a相加)$$

b)P版本设计思路

为了开发P版本就要换另一种思路，经过模拟人日常进行乘法计算得出另一种思路，将a作为被乘数，b作为乘数。第一次取b的0号位置上的数，由于二进制只有01两种可能，结果也只有两种可能：若该为为0，结果为0，若该位为1，结果为被乘数。一次执行完毕后对被乘数进行进位操作，然后再取b的1号位置上的数重复上述操作.....最终经过16次完成乘法。

$$x_4x_3x_2x_1x_0 * y_4y_3y_2y_1y_0 = \sum_{i=0}^{4} y_i * 2^i * (x_4x_3x_2x_1x_0) = \sum_{y_i=1} y_i * 2^i * (x_4x_3x_2x_1x_0)$$

二.设计与优化过程

a) 第一代L版本

```
0001 010 001 1 00000 ;R2=R1
0000 010 00000 0011  ;if R2 == 0,halt
0001 111 111 000 000 ;R7 = R7 + R0
0001 010 010 1 11111 ;R2 = R2 - 1
0000 111 11111 1100  ;jump to line 2
```

b) 第一代P版本

```
0001 110 110 1 00001 ;将R6置1
0101 011 001 000 110 ;取乘数对应数字
0000 010 0000 00111  ;若该位为0，则跳过下面部分，直接进入第18行
0101 101 000 1 11111 ;将R0的值赋给R5，R5为保存经过移位后的被乘数
0101 100 010 1 11111 ;将R2的值赋给R4，R2当作计数器，R4为另一个计数器
0000 010 0000 00011  ;下面进行移位操作，直到R4为0结束
0001 101 101 000 101 ;对R5中的数进行移位
0001 100 100 1 11111 ;R4的值减1
0000 111 1111 11100  ;无条件跳回到断点2
0001 111 111 000 101 ;将R5与R7中的数相加
0001 010 010 1 00001 ;计数器加1
0001 110 110 000 110 ;R6移位
0000 101 1111 10100  ;跳回断点1执行程序
```

经过优化得到第二代P版本

C) 第二代P版本

```
0001 010 010 1 00001;R2 <= 1
0101 011 001 000 010;R3 = R1 AND R2
0000 010 0 0000 0001;if R3== 0, jump to line 5
0001 111 111 000 000;R7 = R7 + R0
0001 000 000 000 000;R0 <<= 1
0001 010 010 000 010;R2 <<= 1
0000 101 1 1111 1010;if R2 != 0, jump to line 3
```

经过优化后的P版本去除了两个计数器（这两个计数器属实是多此一举）大大简化了代码，执行次数与代码行数均得到较大提升，并对注释进行规范化。

三.统计优化前后的性能

- 第一代L版本代码行数为5行。
- 第二代L版本与第一代相同，无改进，仍为5行。
- 第一代P版本执行平均指令条数为297条。

  【指令数统计方法：程序稳定循环16次，若R3为0，执行指令条数为5条，若R3不为0，执行指令条数为32条。则每次循环平均执行指令为18.5条，16次循环执行296条。再加上R2置1这一条指令，共有297条】
- 第二代P版本执行指令p平均条数为89条。

【指令数统计方法：程序稳定循环16次，若R3为0，执行指令条数为5条，若 R3不为0，执行指令条数为6条。则每次循环平均执行指 令为5.5条，16次循环 执行88条。再加上R2置1这一条指令，共有89条】